#### 今日头条Apk瘦身

随着版本迭代，功能增加安装包体积也会慢慢增大。

今日头条576版本APK达到了25M，通过一系列的优化，到目前的607版本为12M。本文主要是介绍头条APK瘦身中用到的一些方法。

## APK分析

既然是要优化APK的大小，那首先就得看下APK文件的构成。

Android Studio在2.2版本添加 APK Analyzer功能，可以直接打开apk文件，如下图所示 image

APK文件主要有如下几部分组成：

 res主要是存放图片资源

 lib主要是存放so库，各个cpu架构

 classes.dex是java源码编译后生成的java字节码文件，因方法数限制拆分了多个dex

 assets主要存放不需要编译处理的文件

 resources.arsc是编译后的二进制资源文件，包括图片、文本索引等

 META-INF 签名信息

 AndroidManifest.xml 描述配置文件

从APK的构成中可以看出占比较大的几个部分，可以着重对其优化

## 优化

### res文件夹

#### 图片资源压缩

1、[ImageOptim](https://imageoptim.com/)

提供了相应客户端，支持通过客户端批量处理，mac上可以使用如下命令开启：

find . -name '\*.png' | xargs open -a ImageOptim

2、[TinyPng](https://tinypng.com/)

没有提供客户端，支持拖拽到网页处理；提供了HTTP API来批量处理，但是有数量限制，每个key每个月可以压缩500张。 开发了一个gradle插件来批量操作，网上也有一些类似的插件：[TinyPng Gradle插件](https://github.com/mogujie/TinyPIC_Gradle_Plugin)

#### 移除无用资源

1、通过使用Lint检测删除无用资源，某些业务代码删除的时候遗漏了相应资源，可以写个脚本检测移除不再使用的资源

2、添加shrinkResources设置项([官方说明](https://developer.android.com/studio/build/shrink-code.html))，有0.18M的优化空间，但是该设置有风险如果要使用需要做好测试

3、选择支持合适的图片，目前有ldpi mdpi hdpi xhdpi xxhdpi xxxhdpi资源文件夹，可根据自己app的用户设备选择支持2-3种即可（当然一套也行） 高版本的gradle已不再支持通过resConfigs "nodpi", "hdpi", "xhdpi", "xxhdpi"配置支持的资源，只能人肉删除。如果你只想打包某一种屏幕密度的资源，可以使用分包策略，添加如下density配置可以只支持打包xhdpi资源（如果出现某些资源xhdpi没有，而其他文件夹包含的情况也不用担心，gradle会保留相应资源），这种配置最终会出多个apk包，具体介绍可参看[官方说明](https://developer.android.com/studio/build/configure-apk-splits.html)。

 splits {
 density {
 enable true
 reset()
 include "xhdpi"
 compatibleScreens 'small', 'normal', 'large', 'xlarge'
 }
}

4、如果想整体移除res下某个文件夹可以添加如下aaptOptions配置，而不用打包时手工删除，多个文件夹用:隔开

aaptOptions {
 ignoreAssetsPattern 'color-night-v8:drawable-night-v8'
}

### arsc文件

resource.arsc文件记录了资源id和资源的对应关系（字符串的内容，图片的相对路径等）

#### 减少语言支持

目前包括各种语言（v7包引入），点击resources.arsc可以看到支持80种image可以通过修改gradle配置，去除不需要部分，这里我们保留4种

defaultConfig {
 resConfigs "zh-rCN", "zh-rHK", "zh-rTW", "en"
}

只保留"zh-rCN", "zh-rHK", "zh-rTW", "en" 减少不必要的语言（80种减到5种，有一个default）apk可减少0.61Mimage

#### 资源混淆

开源解决方案[AndResGuard](https://github.com/shwenzhang/AndResGuard)可以看下，通过使用段路径和压缩可以减小apk，需要注意的是你的项目中某些资源需要keep，减少了1.5M。

### lib文件夹

#### 架构支持

Android系统目前支持以下七种不同的CPU架构：ARMv5，ARMv7 (从2010年起)，x86 (从2011年起)，MIPS (从2012年起)，ARMv8，MIPS64和x86\_64 (从2014年起)

每一个CPU架构对应一个ABI：armeabi，armeabi-v7a，x86，mips，arm64-v8a，mips64，x86\_64

所有的x86、x86*64、armeabi-v7a、arm64-v8a设备都支持armeabi架构的.so文件，x86设备能够很好的运行ARM类型函数库，但并不保证100%不发生crash，特别是对旧设备。64位设备（arm64-v8a, x86*64, mips64）能够运行32位的函数库，但是以32位模式运行，在64位平台上运行32位版本的ART和Android组件，将丢失专为64位优化过的性能（ART，webview，media等等）。所以一般的应用完全可以根据自己业务需求选择使用armeabi或者armeabi-v7a一种支持就行。

可以通过gradle配置

 defaultConfig {
 ndk {
 abiFilter "armeabi"
 }
}

比如：微信、微博、QQ只保留了armeabi，Facebook、Twitter、Instagram只保留了armeabi-v7a

假设只支持了armeabi，如果有特殊要求（比如视频应用）需要用到部分armeabi-v7a的so，可以通过改名放到armeabi文件夹中，根据手机实际情况选择加载。

#### 动态下发

比较大的so可以选择动态下发的形式延迟加载，代码上需要加一些判断逻辑。

### dex文件

1、添加设置minifyEnabled true，混淆、压缩代码，这个设置现在app应该都已经添加了。

2、删除一些无用库，早期为了兼容低版本手机，添加了一些兼容库，随着时间推移APP支持的最低版本也在升高，之前的一些无用库就可以移除。

3、插件下发业务模块 添加插架框架，将部分代码延迟下发加载，这部分效果显著。

### 其他

极端情况下可以考虑以下两种方式，可以优化约1M的空间

#### debug信息

一般我们会配置Proguard保留行号等信息用于线上日志分析，极端情况下也可考虑移除这部分，会有5%-10%的效果，可以减少了0.5M，但是出于方便性暂未移除。

-keepattributes SourceFile,LineNumberTable

#### supportv7包

如果对supportv7包依赖的不多，可以直接把使用到的内容copy出来单独处理，毕竟该包会增加至少0.4M的体积，业务复杂后这部分并不好操作和后续维护，头条暂时并没有使用。

### TODO

功能迭代不止，瘦身事业不息。

要维持和继续减小apk包，必须要不断优化，现在又如下思路还没有实施，可以看下

1、Google的support-v4包新版本已经做了拆分，24.2.0版本拆分成了5个module:support-compat、support-core-utils、support-core-ui、support-media-compat、support-fragment，可以根据自己需要单独引用相应的module。 v7包也会依赖v4，maven依赖有个好处，可以通过exclude单独剔除相应依赖，如下：

 compile ('com.android.support:appcompat-v7:24.2.0') {
 exclude module: 'support-v4'
 }
 compile 'com.android.support:support-fragment:24.2.0'

不过目前各家APP对于support包的使用较深，support包各模块也会有相关依赖关系，具体能不能使用还需要看实际情况了。

2、使用[ReDex](https://github.com/facebook/redex.git)优化，这是Facebook开源的一个减小安卓app大小以提高性能的工具，集成的话有风险需要多测试，[教程](https://code.facebook.com/posts/998080480282805/open-sourcing-redex-making-android-apps-smaller-and-faster/)。

3、减少java隐藏开销，比如一些自动生成的函数等。